The Entity Registry System: Collaborative Editing of Entity Data in Poorly Connected Environments

Christophe Guéret
DANS, Royal Dutch Academy of Sciences
The Netherlands
{firstname.lastname}@dans.knaw.nl

Philippe Cudré-Mauroux
eXascale Infolab, University of Fribourg
Switzerland
{firstname.lastname}@unifr.ch

Abstract

There are about 4.5 billion people in the world who have no or limited Internet access. Those are deprived from using entity-driven applications that assume data repositories and entity resolution services are always available. In this paper, we discuss the need for a new architecture for entity registries. We propose and evaluate a new general-purpose Entity Registry System (ERS) supporting collaborative editing and deployment in poorly-connected or ad-hoc environments. The reference open-source implementation is evaluated for scalability and data-sharing capabilities.

Introduction

Quoting Tim Unwin, it can be observed that "information and knowledge have always been central to the effective functioning of human societies" (Unwin 2009). Over the past decades, computational systems greatly influenced the scale and span of information sharing. We are entering an era where computer-mediated interactions and, soon, machine-to-machine interactions ("Internet of Things") are the norm.

Yet, although human society as a whole has its activities centred around information and knowledge not everyone benefits from the same tools. A digital divide is increasingly observed between the few having access to information and communication technologies (ICT) infrastructures and those who don't. As of 2012, an estimated 65 % of the world's citizens are still deprived from Internet access[1].

Enabling data sharing for disconnected populations is a goal that encompasses several key societal, economical and technological challenges (Becker 2004; Guéret et al. 2011). Among these, the typical assumption that connectivity to data repositories and entity resolution services are always available make many data-driven applications out of reach. These applications may have a critical impact on the life of the 4.5 Billion people that are off-line, but are inaccessible to them due to the architecture of today's data registries. There is a need for novel approaches for the collaborative editing of entity data in poorly connected environments.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] http://www.internetworldstats.com/stats.htm, last visited June 2014

Example use-case

The Sugar Learning Platform [2] originally developed for the XO laptop of the One Laptop Per Child initiative (OLPC) [3] is an education environment aimed at young learners. Sugar lets users perform a number of game-oriented activities such as collaborative text writing, photo-novel editing and abacus counting. Everything a kid does with his laptop gets logged into the "Journal", a diary used as a tool for self-reflection and to get guidance from teachers who can consult it [4]. There is today a need to extend the usage of the Journal data to collect more general statistics on the learning process of the users. Some solutions have been already put in place (Verma 2014), though these are either using dumps of the entire Journal contents or require an external server to export the data. This poses threats to privacy (Guéret 2014) and limits usability when connectivity is a challenge. In the following, we argue for a new kind or data repository that can handle the private content of the Journal as well as exposing public statistics about its contents. This system must be able to work in very different connectivity contexts and ensure that at least local teachers always get access to the public data.

Research questions and main contributions

In this work, we pursue the design of a generic information system that is able to store and share structured data. This data consists in descriptions associated to uniquely identified entities. The related research questions for the design of such a system are:

- What should be the resolvability, consistency, content relevancy and autonomy capabilities of systems designed for collaborative editing of entity data?
- Can decentralised approaches by used in poor connectivity contexts and if so at which cost?

We tackle these points with a conceptual and practical approach, sketching out a general information model for decentralised entity registries and providing a concrete reference implementation. More specifically, our main contributions are:

[2] http://sugarlabs.org/, last visited June 2014

[3] http://one.laptop.org/, last visited June 2014

[4] http://wiki.laptop.org/go/Journal_Activity, last visited June 2014

- A novel hybrid architecture for the collaborative editing of entity data;
- A reference implementation of the Entity Registry System (ERS);

Outline

This paper is based on earlier introduction and early prototype description of ERS (Guéret and Cudré-mauroux 2014; Charlaganov et al. 2013b). It is structured as follows: we first give an overview of common collective knowledge curation architectures and discuss them in the light of our example use-case. Subsequently, we give a description of the Entity Registry System and of the reference implementation we propose before concluding the paper.

Background and related work

Emerging on-line applications based on the Web of Objects and Linked Open Data typically assume that connectivity to data repositories and entity resolution services are always available. Yet, there are 4.5 Billion individuals world-wide who are deprived from stable Internet access and for which common architectures for entity registries can not be deployed effectively. In what follows, we review and point out the limitations of current entity registries when deployed in poorly connected environments.

Common architectures for entity registries

We can observe three main approaches to design a repository for semi-structured entity data: centralised, hierarchical and distributed.

Centralised The Linked Data movement [5] has been pushing towards publishing and interlinking public data in standard formats, which enables the automated discovery, management and integration of structured resources online. The adopted technology is based on HTTP URIs and RDF. The resolution of an entity given its identifier boils down to three steps in that context:

1. discovering the IP address where the HTTP URI is supposed to be hosted (for example using the Domain Name System—DNS)
2. contacting the corresponding server and negotiating the content (*e.g.*, to serve an HTML version of the RDF data if the client is a Web browser)
3. retrieve the structured description of the entity over HTTP.

This process is commonly called entity dereferencing since it is similar to general URI dereferencing on the Web [6]. In the context of the Okkam project, the Entity Name System (ENS) (Bouquet, Stoermer, and Bazzanella 2008) has been proposed. It is defined as a service to resolve entity names to their global identifiers (called Okkam IDs). This is made available thanks to a repository of entity profiles described as a set of attribute-value pairs, and a mix of matching components that select the correct identifiers for an entity request which may be submitted in the form of a structured (*i.e.*, attribute-value) or unstructured (*i.e.*, keyword) query.

Hierarchical The Domain Name System (DNS) used on the Internet to resolve domain names to their corresponding IP addresses works in a hierarchical manner. The top of the domain name space is served by so-called root name servers, pointing to name servers maintaining authoritative information for the top-level domains (*a.k.a.* "TLDs", such as ".ch" or ".com"), which in turn point to second-level domains (*e.g.*, "unifr.ch") and so on and so forth until the last iteration, which returns the IP address of the query[7]. Though originally not designed for this purpose, it is be possible to extend the current DNS infrastructure to create a full-fledged entity registry. In that context, we recently suggested an extension of the DNS (Cudré-Mauroux et al. 2011) to serve authoritative meta-data about Internet domains, levering both the DNS Text Record field (DNS TXT) and new cryptographic features (DNSSEC).

Distributed Distributed Hash-Tables (DHTs), such as Chord (Stoica, Morris, and Karger 2001) or our P-Grid system (Aberer and Cudré-Mauroux 2003), provide distributed, scalable hash-table-like functionalities that can be used to store entity identifiers as well as related meta information in ad-hoc environments. Through dynamic load-balancing and replication, those networks provide fault-tolerance and efficient networking primitives where arbitrary requests can typically be resolved in $O(log(N))$ messages, where N is the number of nodes in the P2P network, from any entry point to the network. GridVine (Cudré-Mauroux, Agarwal, and Aberer 2007) uses such a DHT to locate nodes responsible for queries related to a specific entity. Swarm intelligence and bee computing can also be used in this context to provide an even more flexible alternative to DHTs (see S4 (Mühleisen et al. 2010) and DataHive (Kroes, Beek, and Schlobach 2013)). The "divide and conquer" hierarchical computing paradigm can also be applied to storing and retrieving the descriptions of entities. Illustrating this, the cluster-oriented database system HBase has recently been used to store RDF data (Sun and Jin 2010; Fundatureanu 2012). The interested reader is referred to (Mühleisen, Walther, and Tolksdorf 2011) and (Cudré-Mauroux et al. 2013) for a more exhaustive study on distributed storage model and performances of distributed RDF storage.

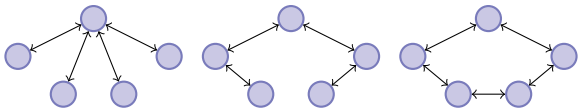
Discussion

Figure 1 provides a visual summary of the different architectures. It can be observed that the centralised approach creates a strong dependency on a central node used to store and serve the content (see Figure 1a). The single point of failure and the scalability issues that are associated can be mitigated by a hierarchical approach (see Figure 1b) whereas the distributed architecture more equally shares the load among all the nodes in the network (see Figure 1c).

[5] http://linkeddata.org/, accessed June 11, 2014

[6] http://www.w3.org/2001/tag/doc/httpRange-14/2007-05-31/HttpRange-14, accessed June 11, 2014

[7] In practice, domain names are often cached at various levels, for instance at the client-side, or at the level of the DNS server provided by the Internet Service Provider in order to limit the load



(a) Centralised (b) Hierarchical (c) Distributed

Figure 1: Overview of the interactions among nodes for centralised, hierarchical and distributed systems for the storage and sharing of entity data

These different common approaches have to be considered in the light of our example use-case, which is to support knowledge sharing in poorly connected environments. The following criteria can be defined based on our use-case:

Resolvability: finding the information associated to an identifier;

Consistency: dealing with inconsistent updates inserted by different users;

Relevancy: caring for context dependant information needs;

Autonomy: amount of dependency set among all the nodes in the system.

The result of matching of the above architectures against these criteria is visible in Table 1. We observe that none of the discussed approaches commonly found in the literature fits our context. Centralised approaches require a regular access to a central store that cannot be guaranteed in environments where connectivity is an issue. The centralisation of information can also be sensitive for knowledge collected in the context of political unrest. Hierarchical architectures are doing better but depend on the availability of a number of nodes in the tree to resolve the description of entities. Going distributed is the most promising way, though this comes at a cost on consistency and resolvability and sets a stronger dependency among the nodes.

	Centralised	Hierarchical	Distributed
Resolvability	+	-	o
Consistency	+	o	-
Relevancy	-	+	o
Autonomy	-	o	+

Table 1: Overview of the advantages of the three common architectures in light of the criteria introduced. A + indicates a fit of the approach, a - signifies that the architecture does not fit and a o is used for cases needed specific implementations.

The design goals of the Entity Registry System (ERS) hereafter introduced is to combine a decentralised and hierarchical approach into a flexible system that can be used off-line and on-line with low resources. For this, we adopt a fully decentralised approach with very loose dependencies among the nodes.

on authoritative DNS servers.

The Entity registry system

With ERS, we aim to provide a local system that can also operate in connected settings. In our approach, our system is "off-line by default" but it is able to seamlessly transition from off-line to on-line settings depending on the current connectivity.

Information model

ERS stores semi-structured descriptions of entities to power data-intensive applications and is designed around lightweight components that collaboratively support data sharing and editing in intermittently connected settings. It is compatible with the RDF data model and makes uses of both Internet and local networks to share data, but does not base its content publication strategy on the Web. No single component is required to hold a complete copy of the registry. The global content consists of the union of what every component decides to share.

Documents Statements expressed by a user about an entity are wrapped into unique logical containers hereafter referred to as a "documents". Every set of statements about a given entity contributed by one single author is stored in a unique document. Every document gets assigned a unique identifier within the system, a version number and a link to its author. Because every document is internally consistent, potential conflicting statements are necessarily found in different containers thereby avoiding consistency issues. This system also facilitates tracking the provenance of all the individual contributions made to the registry. The identifier for the entities can be freely chosen by developers using ERS. It is however advised to use a URN `urn:ers:<path>:<identifier>`, with "path" being a ontological path and "identifier" an identifier of the entity. For example, `urn:ers:pet.feline.cat:felix`. Part of the goals of the project is to investigate how some URNs end up being preferred over others as part of a collaborative reinforcement process similar to the one driving data sets re-use on the LOD (Maali, Cyganiak, and Peristeras 2011). Every node running ERS can freely contribute to the description of any entity by creating documents extending its description.

Stores As highlighted in the introduction, one may want to contribute to the description of an entity with parts of the description that are private and others that are public. The former being meant for publication & consumption on a single machine and the later being freely shared among the peers. The need to temporarily store some documents produced by some of these peers may also arise. We cover those three different needs by implementing three different stores in each of the ERS nodes (see Figure 2). The way documents are exchanged among the nodes across their different data stores depends on their type. The three types of nodes and their interaction are hereafter introduced.

Example As an example let us consider structured data associated to an entry of an activity journal. We will hereafter consider the journal of the educative environment "Sugar".

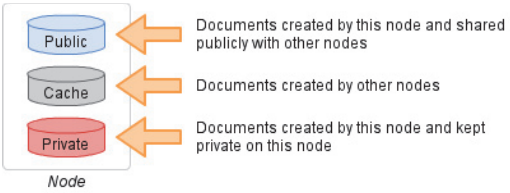


Figure 2: Every node in the system has three different data stores: one for his own public contributions to the knowledge base, one for that of others and one for additional content that is private

Table 2 shows part of such a journal entry. The table contains the unique identifier associated to this entry ("uid"), the title given by the user ("title"), the name of the activity associated to the record ("activity"), the timestamp of the last update ("timestamp") and the color of the icon showed in the Journal viewer ("icon_color").

Key	Value
uid	c7022a25-892a-4848-b539-8b525371d571
title	Record Activity
activity	org.laptop.RecordActivity
timestamp	1375588726
icon_color	#FF8F00,#BE9E00

Table 2: Excerpt of a Journal entry showing some of the most important values

The JSON data showed in Listing 1 is the matching document for Table 2 as used in ERS. For the encoding, we had the choice between using predicates as keys or using two synchronised arrays of key/value pairs (Charlaganov et al. 2013a). After some testing and taking inspiration from related work (LD-In-Couch [8], JSON-LD [9]), we opted for using predicate as keys. It can be observed that a field has been added to keep track of the owner of the document (`@owner`) along with other additional fields recommended for JSON-LD description of entities.

Listing 1: Private journal entry serialized using predicates as keys

```
{ "@context": { "ers": "urn:ers:meta:vocabulary" },
  "@id": "urn:ers:journal-entry:c7022a25",
  "@type": "ers:JournalEntry",
  "@owner": "urn:ers:host:100953e1",
  "ers:title": "Record_Activity",
  "ers:activity": "org.laptop.RecordActivity",
  "ers:timestamp": "1375588726",
  "ers:icon_color": "#FF8F00,#BE9E00" }
```

The content of a Journal entry is inherently private and will be stored into the matching document store. Our motivating use-case and running example require to also publish a public document to be used to collect statistics over the usage of the laptop. This is ensured by creating another document that will be placed in the public store. The document for Listing 1 is shown in Listing 2.

Listing 2: A public document exposing some data about the journal entry

```
{ "@context": { "ers": "urn:ers:meta:vocabulary" },
  "@id": "urn:ers:stats:org.laptop.RecordActivity",
  "@type": "ers:JournalStatEntry",
  "@owner": "urn:ers:host:100953e1",
  "ers:journal_entry":
        "urn:ers:journal-entry:c7022a25"
  ,
  "ers:activity": "org.laptop.RecordActivity"}
```

The most interesting thing to observe in Listing 2 is that all the statistics entries are designed to describe the same entity (in this case "urn:ers:stats:org.laptop.RecordActivity"). Every individual log contributes to the description of a single entity associated with an activity by adding links to journal entries. Though the entries are not public, their number per user can be counted and depicted as usage statistics.

Node types

ERS is articulated around three types of components: Contributors, Bridges, and Aggregators. The components can be deployed on any kind of hardware ranging from low-cost computing devices such as RaspberryPis [10] to data centres.

Contributors use and create the contents of the registry. They create and delete entities, look for entities, and contribute to the descriptions of the entities.

Bridges do not contribute to the contents of the registry. They are used to enable asynchronous communication among Contributors found on a same network and connect isolated pools of Contributors located in different closed networks. Bridges store documents only for a limited amount of time (*e.g.*, using *soft states*) due to their limited capacity, focusing on keeping the most recently used documents.

Aggregators are used to gather a copy of the documents transiting through the Bridges. When needed, an Aggregator can provide a single entry point to the global content of an ERS deployment. An Aggregator may also expose the aggregated contents to other systems, for instance to the Web of Data.

Contributors are the primary component necessary for the operation of ERS. Bridges and Aggregators are optional and may be deployed depending on the use-case. Looking at the overall architecture of ERS in the light of P2P systems, one could see Bridges and Aggregators as supernodes. Their role is however slightly different. Unlike common P2P architectures where supernodes are regular nodes performing the same features with more computation power the Bridges and Aggregators are not akin to Contributors. They are only needed to provide aggregation of content and asynchronous communication among the Contributors. From a requirement point of view their usage can contribute to tackling

[8] https://github.com/mhausenblas/ld-in-couch
[9] http://www.w3.org/TR/json-ld/
[10] http://www.raspberrypi.org/, accessed December 9, 2013

the utility/validity gap observed when contextualising global data with local data (Raftree 2013).

Synchronization schema

The interaction between Contributors, Bridges and Aggregators is designed to maximise the data flow while limiting the amount of data stored by each node. An overview of the interactions is given in Figure 3. We hereafter give more detail about the pair-wise mode of communication.

Contributor-Contributor Contributors can exchange updates about the documents they have cached. An application using ERS can also choose to cache documents found in a peer Public or Cache document store.

Contributor-Bridge The basic interaction process between a Contributor and a Bridge is simple and consists in i) sending to the bridge every new document from the Contributor's public store and ii) exchanging updated versions of cached documents. In addition, Contributors can query the Bridges for data and persist in their cache the retrieved documents.

Bridge-Bridge Bridges exchange updates for the documents found in their cache document store. Additionally, a Bridge can query another Bridge for "interesting" documents and choose to cache the results to make them available to the Contributors it typically interacts with. In our implementation, interesting documents are those containing a statements about an entity already found in the cache.

Bridge-Aggregator Bridges push new documents from their cache to the cache of Aggregator. The two nodes also exchange updates for the documents they already have in their cache.

Looking at Figure 3 it can be noted that the content of the private store is never shared. Furthermore, as the Bridge and Aggregator are not considered to directly contribute to the common knowledge, their public stores are typically not used. Deletion of documents are considered as an updated status that also gets replicated. An entity ceases to exist when there is no more document containing a statement expressed about it. In addition to these automated synchronisations, an application using ERS can search for entities described by other peers in the network and then can eventually persist the results retrieved.

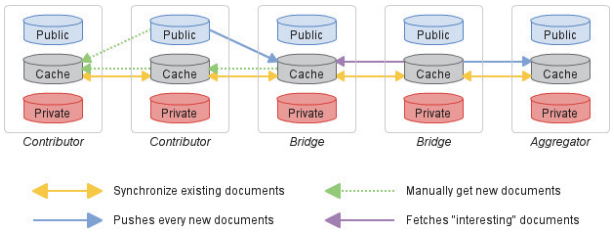


Figure 3: Synchronization schema among the different type of nodes in the system. The solid arrows are automated processes, dashed transfers are triggered by applications using ERS

Discussion

We introduced earlier four criteria to study the suitability of common architectures for knowledge information systems to our specific context. In what follows, we discuss how ERS matches the different criteria.

Resolvability: an entity can be resolved at any point in time by fetching documents from all the contributors and bridges found in the network neighbourhood and on the stores of the component emitting the query. The entity always resolves but the degree of completeness of the description depends on the spread of the different documents as well as the global connectivity;

Consistency: every contribution is wrapped onto a self-consistent document that can only be updated by its author. Conflicting statements describing an entity are necessarily found in different documents, making it possible to defer the conflict resolution to the application consuming the aggregated description of the entity;

Relevancy: contributors and bridges found in the network neighbourhood are the only additional source of information one component may query. This gives a natural priority to data found in a close network proximity. When combined with a mesh networking, this logical proximity gets a physical dimension. Data coming from further away has to transit through bridges;

Autonomy: a deployment made of two contributors is fully functional and enables the collective description of entities. The usage of bridges and aggregators is fully optional.

Reference implementation

In this section, we give a detailed overview of the reference implementation of ERS. All of the concepts we mention in this paper are implemented and available as open-source code from the "ers-devs" group on GitHub[11].

Contributors and Bridges

Contributors and Bridges are implemented using the same code designed to run on small hardware under sporadic connectivity contexts. The operating mode of an ERS node is set in a configuration file. We choose to use CouchDB[12] to persist all the data locally and perform the different synchronizations. The main advantage of CouchDB is that it provides built-in mechanisms for flexible replication of the data. The CouchDB replication system can be configured to match our needs (*e.g.,* to create triggers and filters based on the entity descriptions).

Data storage Internally, CouchDB defines *documents* that are used to store and replicate payloads of JSON data. We decided to use these documents as a one-to-one equivalent to our notion of document introduced earlier. That is, every document in a CouchDB database correspond to a set of statements a Contributor made about a specific entity. The

[11] http://ers-devs.github.io, accessed December 6 2013
[12] http://couchdb.apache.org/

only difference with the code of Listing 1 is the addition of two "_id" and "_rev" fields that CouchDB uses internally to uniquely track and version the documents it stores. Similar to the notion of named-graphs in RDF the "_id" differentiates two document which could otherwise look the same. "_rev" is used to track revision of a specific documents.

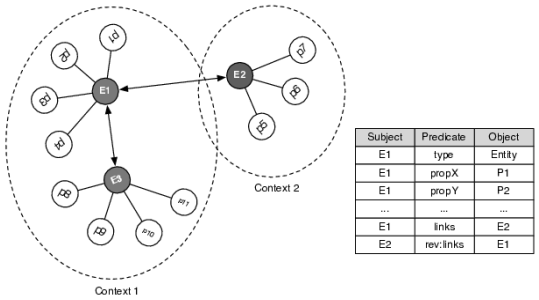
Peers discovery Contributors and Bridges run a daemon that takes care of exposing the list of connected peers running ERS via the ERS API. This list of peers is used internally by ERS to dynamically configure the synchronization rules of CouchDB and take care of the automated propagation of updated documents (solid arrows on Figure 3). The daemon relies on the ZeroConf protocol to announce itself on the network and discover other running processes. We currently support only the Avahi implementation of this protocol commonly used on the GNU/Linux operating systems.

Operating mode The switch between the Contributor and Bridge operating mode is done via a configuration file. By default, every ERS node is a Contributor. As highlighted earlier, the main difference between a Contributor and a Bridge is that the later does not contribute contents using its public store. The synchronization rules are also different as Bridges have two core functionalities: distribution of local data in semi-connected networks and synchronization to other closed networks.

Aggregators

While Contributors and Bridges are designed around sporadic or ad-hoc connectivity, an Aggregator is an optional, always-on component. A single instance of an Aggregator might not always store the entirety of data, but can store references to other servers providing such information, thereby acting like a typical registry for non-local content. Due to the distributed nature of how data is processed, cached and shared in ERS, providing a consistent storage layer is not trivial. To achieve the desired consistency, we implemented a transaction layer on top of our data store. Using traditional distributed transaction schemes from relational database management systems is out of scope for two reasons: first, allowing multi-side distributed transactions would severely slow down the overall performance, and second, even if the Aggregator is considered to be always available, this is not true for the Bridges sending it data. Thus, the goal must be to achieve the best possible throughput on slow dial-up connections even if the connection is dropped multiple times.

Data model The overall data model of the entity registry is shown in Figure 4. There are two important semantic properties that are visible in this figure. First, the properties of specific entities have no connections between each other. This means that for a given triple $t = \{s, p, o\}$ with $s \in S, p \in P, o \in O$ there can be no two triples that share the same predicate and object. This effectively describes a uniqueness constrain on all triples on the p and o attributes. The second semantic property is that modifications on the graph of entities are typically separated by the context in which they operate, making it less likely that collisions between different contexts will occur.



Subject	Predicate	Object
E1	type	Entity
E1	propX	P1
E1	propY	P2
...	...	...
E1	links	E2
E2	rev:links	E1

Figure 4: Entity Model Overview

Concerning links, only connections between entities are allowed. For simplicity, a connection between an entity is seen as bidirectional. On the storage, side we model this as two entries in the graph, one describing the original connection and a second entry describing the inverse relation. This allows to navigate the path in both ways and to keep the original semantics about how the connection was created.

Data storage The foundation for the storage layer of the Aggregator is a cluster of Apache Cassandra nodes with an additional abstraction layer for storing RDF inside the cluster based on CumulusRDF (Ladwig and Harth 2011). Basing the Aggregator on Cassandra allows us to minimize the entity look-up latency, while also being able to dynamically scale-out to multiple machines to aggregate very large amounts of entity data whenever necessary. While Cassandra natively offers no support for transactions or atomic operations on data stored in the cluster, we implemented some specific support for atomic operations at the RDF layer.

Transactions Inside our thin application layer sitting on top of Cassandra, we define the following atomic operations: `Insert entity (IE)`, `Insert property of entity (IP)`, `Update property of entity (UP)`, `Delete property of entity (DP)`, `Delete entity (DE)`, `Shallow entity copy (SC)`, `Deep copy of an entity (DC)`, `Insert bi-directional link between two entities (IL)`, `Delete link between two entities (DL)`.

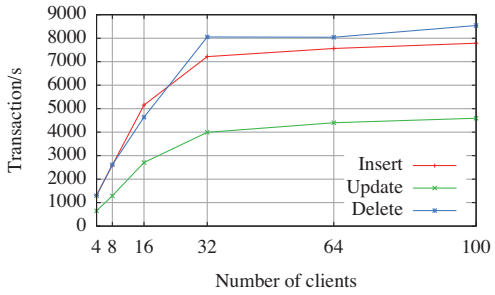
Since transaction support must be implemented on a higher level, we define a multi-level locking scheme that allows hierarchical locking of the different elements of an entity. In contrast to traditional relational databases, our locking approach has the possibility to lock an entity even if it does not exist by referencing the unique ID of the entity in our lock table. This allows a strict serialization of conflicting operations, even for insertions. The two hierarchical locks are: L_{E+P} and L_E. The former locks the entity ID and the property, while the latter locks the complete entity. While two L_{E+P} locks can be compatible in case they differ in one of the two parts, two L_P locks are not compatible. Table 3 shows the compatibility for all kinds of operations with

our various types of locks. Since all property locks are compatible, we can achieve a high throughput for most of the incoming operations. For the two lock types, we match the following operations. Using the fine granular L_{E+P} we can run the following operations: IP, UP, DP, SC, IL, DL. Using the L_E lock we can execute: IE, DE, DC. For links and shallow copies the matching property is either `sameAs` or `linksTo`.

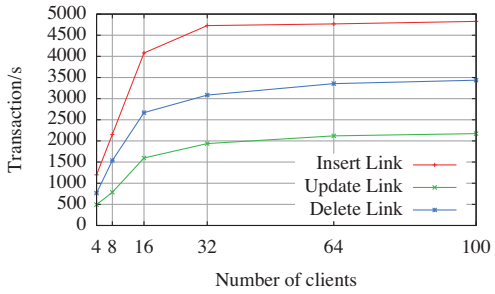
	$L_{E_a+P_c}$	$L_{E_a+P_d}$	$L_{E_b+P_c}$	$L_{E_b+P_d}$	L_{E_a}	L_{E_b}
$L_{E_a+P_c}$	×	✓	✓	✓	×	✓
$L_{E_a+P_d}$	✓	×	✓	✓	×	✓
$L_{E_b+P_c}$	✓	✓	×	✓	✓	×
$L_{E_b+P_d}$	✓	✓	✓	×	✓	×
L_{E_a}	×	×	✓	✓	×	✓
L_{E_b}	✓	✓	×	×	✓	×

Table 3: Operation Compatibility

Performance We experimentally test the performance of atomic operations introduced in Section 3 with 5 servers (8-cores i7 Intel CPU, 8GB total RAM memory, Gigabit Ethernet, Linux kernel 3.2.0, Java SE 1.6), all of them running a Cassandra instance with replication level 2 and having a varying number of clients (2-64). The performances of the Aggregator are reported in Figure 5. We differentiate between basic operations that modify properties of existing entities and linking operations that insert a bidirectional link between two existing entities. Delete operations have the highest throughput as they only require simple marking of a record in the cluster with a tombstone by Cassandra. For linking the situation is slightly different as it requires to bundle two operations—two inserts—in one atomic operation, adding additional overhead.



(a) Basic Operations



(b) Linking Entities

Figure 5: Throughput for different Aggregator operations. All transactions per clients are executed sequentially, but all clients run in parallel.

Conclusions

In this work, we outlined the limitations of entity registries in poorly connected environments, proposed an innovative architecture for a decentralised registry (ERS), and described a reference implementation. We used the practical use-case of gathering and sharing public and private data from the Sugar learning environment. The performance evaluation done on real test-cases and different hardware configurations shows that the system performs well and provides the expected features. Future work will include getting ERS to be adopted for the main-stream version of Sugar and deploying wider instances of ERS.

Acknowledgment This work was supported by the Verisign Internet Infrastructure Grant program.

References

Aberer, K., and Cudré-Mauroux, P. 2003. P-Grid: a self-organizing structured P2P system. *ACM SIGMOD* 32(5005):29–33.

Becker, K. F. 2004. The Informal Economy - fact finding study. Technical Report March, Swedish International Development Cooperation Agency. http://www.eldis.org/go/home&id=15832&type=Document.

Bouquet, P.; Stoermer, H.; and Bazzanella, B. 2008. An entity name system (ens) for the semantic web. In Bechhofer, S.; Hauswirth, M.; Hoffmann, J.; and Koubarakis, M., eds., *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, 258–272. Springer.

Charlaganov, M.; Cudré-Mauroux, P.; Dinu, C.; Guéret, C.; Grund, M.; and Macicas, T. 2013a. The Entity Registry System: Implementing 5-Star Linked Data Without the Web. *arXiv preprint*.

Charlaganov, M.; Cudré-mauroux, P.; Guéret, C.; Grund, M.; and Macicas, T. 2013b. Demonstrating The Entity Registry System : Implementing 5-Star Linked Data Without the Web. In *Proceedings of ISWC2013, Poster and Demo track*, 4.

Cudré-Mauroux, P.; Agarwal, S.; and Aberer, K. 2007. Gridvine: An infrastructure for peer information management. *IEEE Internet Computing* 11(5).

Cudré-Mauroux, P.; Demartini, G.; Difallah, D. E.; Mostafa, A. E.; Russo, V.; and Thomas, M. 2011. A Demonstration of DNS^3: a Semantic-Aware DNS Service. In *ISWC 2011*.

Cudré-Mauroux, P.; Enchev, I.; Fundatureanu, S.; Groth, P.; Haque, A.; Harth, A.; Keppmann, F.; Miranker, D.; Sequeda, J.; and Wylot, M. 2013. NoSQL databases for RDF: An empirical evaluation. In *The Semantic Web ISWC 2013*, volume 8219 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 310–325.

Fundatureanu, S. 2012. A scalable rdf store based on hbase. https://archive.org/details/ScalableRDFStoreOverHBase.

Guéret, C., and Cudré-mauroux, P. 2014. The Entity Registry System: Publishing and Consuming Linked Data in Poorly Connected Environments.

Guéret, C.; Schlobach, S.; Boer, V. D.; Bon, A.; and Akkermans, H. 2011. Is data sharing the privilege of a few ? Bringing Linked Data to those without the Web. In *Proceedings of ISWC2011 - "Outrageous ideas" track, Best paper award*, 1–4. Best paper award.

Guéret, C. 2014. Impressions from sugarcamp3. http://worldwidesemanticweb.org/2014/04/16/impressions-from-sugarcamp3/.

Kroes, P.; Beek, W.; and Schlobach, S. 2013. DataHives: Triple Store Enhancement Using A Bee Calculus. In *Proceedings of the 25th BNAIC*.

Ladwig, G., and Harth, A. 2011. CumulusRDF: Linked Data Management on Nested Key-Value Stores. In *Proceedings of the 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011)*. -.

Maali, F.; Cyganiak, R.; and Peristeras, V. 2011. Reusing cool uris: Entity reconciliation against lod hubs. In Bizer, C.; Heath, T.; Berners-Lee, T.; and Hausenblas, M., eds., *LDOW*, volume 813 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Mühleisen, H.; Augustin, A.; Walther, T.; Harasic, M.; Teymourian, K.; and Tolksdorf, R. 2010. A self-organized semantic storage service. *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services - iiWAS '10* 357.

Mühleisen, H.; Walther, T.; and Tolksdorf, R. 2011. A survey on self-organized semantic storage. *International Journal of Web Information Systems* 7(3):205–222.

Raftree, L. 2013. Open data, open government and critical consciousness.

Stoica, I.; Morris, R.; and Karger, D. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM* 1–45.

Sun, J., and Jin, Q. 2010. Scalable rdf store based on hbase and mapreduce. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 1, V1–633–V1–636.

Unwin, T. 2009. *ICT4D: Information and communication technology for development*. Cambridge University Press.

Verma, S. 2014. The quest for data. http://www.olpcsf.org/node/204.